제13권 제1호
Vol. 13 No. 1

2011

한국 소프트웨어공학 학술대회 논문집(안)

(Preliminary)

Proceedings of 2011 Korea Conference on Software Engineering

- 일시 : 2011년 2월 9일(수)~11일(금)
- 장소 : 한화 휘닉스 파크(강원도 평창)

주최 : 한국정보과학회, 한국정보처리학회

주관 : 한국정보과학회 소프트웨어공학 소사이어티
한국정보처리학회 소프트웨어공학 연구회
한국전자통신연구원

후원 : 삼성 SDS, 비트컴퓨터, (주)모아소프트, (주)이웨이파트너즈, 단국대학교SERC(소프트웨어공학연구센터), 고려대학교 고신뢰 융합소프트웨어 연구센터, 서강대학교 SW 요구 및 검증공학기술 연구센터, 숭실대학교 모바일서비스 SW공학센터, 포항공대 융합소프트웨어개발 연구센터

특별강연 II

D1: 소프트웨어 프로덕트 라인 공학

D2: 임베디드 소프트웨어 시험 및 검증 II

E1: 서비스 지향 아키텍처

테스트 유닛을 이용한 요구사항 기반 테스팅

안성빈*, 박보경*, 김동호*, 서채연*, 김영철**

홍익대학교 컴퓨터정보통신
소프트웨어공학전공
{ahn*, bk*, ray*, seo*, bob**} @selab.hongik.ac.kr

요약: 소프트웨어 테스팅은 소프트웨어 개발 프로세스에서 빼놓을 수 없는 중요한 작업이다. 하지만 부족한 시간과 비용, 인력 때문에 실질적으로 모든 테스트 케이스를 수행하는 것은 불가능하다. 이와 같은 문제점을 해결하기 위해, 본 논문에서는 Use Case approach 에서의 기본 테스트 케이스 추출과 세분화을 위해, 테스트 유닛(Test Unit)를 정의하고, 이를 이용한 요구사항 기반 테스팅을 제안한다. 즉, 유스케이스 패러다임 상에서 요구사항 기반 테스팅 방법이다. 이는 테스트 유닛(Test Unit)기반 테스트 케이스 추출 및 레벨화 기법이다. 최소 테스트 케이스(minimal test case)로 기존의 coverage 를 충족하는 효과를 얻으려고 한다. 이를 위해 본 논문에서는 ATM 기기에 대한 사례를 보인다.

핵심어: 테스트 케이스, 테스팅, UML2.0, 테스트 유닛, 유스케이스 패러다임

1. 서론

최근의 임베디드 시스템은 정보 가전, 정보 단말 등은 물론이고 산업제어기기, 로봇, 사무자동화, 빌딩 자동화, 산업 자동화, 군사, 통신, 물류/금융, 자동차/운송장비, 의료, 게임, 항공관제 등 아주 폭넓고 다양하게 사용된다. 이런 임베디드 시스템 환경은 네트워크 발달과 하드웨어 다양화로 소프트웨어의 규모가 커져가고 있다[1]. 이런 규모의 변화로 소프트웨어 테스팅은 하드웨어보다 시간과 노력, 인력이 많이 소비된다[2].

이런 상황에서 소프트웨어의 품질은 중요한 의미를 가진다[3]. 최종으로 완성된 제품이 올바르게 테스팅 되지 못하면 금전적인 손실뿐만 아니라 시간 낭비, 비즈니스의 이미지 손상, 그리고 인명피해까지 발생시킬 수 있다. 그래서 소프트웨어의 품질 향상을 위해 소프트웨어 개발 전 테스팅을 통해 오류를 찾는 요구사항 기반 테스팅이 필요하다[4]. 일반적인 테스트는 모든 개발이 끝난 다음 테스트를 실행한다 하지만 오류의 대부분은 개발초기에 프로젝트 참가자 간 의사소통의 문제로 발생한다[5]. 결국 대부분의 오류는 잘못된 요구사항으로부터 시작되는 것이다. 또한 잘못된 요구사항은 개발단계가 진행될수록 그것을 수정하는 비용이 기하급수적으로 늘어나기 때문[6]에 요구사항 단계부터 많은 노력을 할 필요가 있다. 요구사항 기반 테스팅은 이런 문제를 반영한 것이다. 즉, 개발 후에 테스팅 하는 것이 아니라 요구사항 단계부터 테스팅을 진행하여 개발 초기에 발생할 수 있는 문제를 미리 발견하고 수정하는데 목적이 있다.

실제로 소프트웨어 테스팅에서 모든 테스트 케이스를 수행하는 것은 불가능하다. 그것은 부족한 시간과 비용, 인력 때문이다[7]. 또한 테스팅을 진행하기 위해서는 새로운 인력 보충보다는 기존에 인원 교육을 해야 한다. 그렇기 때문에 복잡하고 어려운 테스팅은 사용할 수 없다.

이를 위해 본 논문에서는 기존의 유스케이스 패러다임에서 디자인 유닛[10]을 정의하고, 그를 통한 테스트 계획을 적용한 아이디어를 기반으로, 테스트 유닛의 요구사항 기반 테스팅을 제안한다. 제안한 방법은 다음과 같은 특징을 갖는다. 첫째, 제안한 테스트 유닛에 대한 정의 부분이다. 이를 기반으로 테스트 케이스 템플릿(template)들의 생성을 통해 테스트 케이스를 생성한다. 둘째, 생성된 테스트 케이스의 레벨화이다. 앞서 정의된 테스트 유닛을 통해 세부적인 테스트 케이스에 대한 레벨 설정으로부터 효율적인 테스팅을 할 수 있다.

본 논문의 순서는 다음과 같다. 2 장에서는 관련연구로써 요구사항 기반 테스팅에 대해서 기술한다. 3 장에서는 테스팅 유닛을 정의하고, 그에 대한 테스트 케이스 템플릿을 보인다. 또한, 효율적인 테스팅을 위한 테스트 케이스 레벨화에 대한 구체적인 방법에 대해서 언급한다. 4 장에서는 적용사례로 ATM 에 대한 테스팅 방법을 실제적인 테스트 케이스를 통해서

이 논문은 2010 년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업(2010-0012117)과 교육과학기술부와 한국연구재단의 지역혁신인력양성사업으로 수행된 연구결과임.

보인다. 5 장에서는 결론 및 향후 연구에 대해서 기술한다.

2. 관련연구

2.1 요구사항 기반 테스팅

요구사항 기반 테스트는 다음과 같은 7 개 활동에서 나누어질 수 있다[8]:

1. 테스팅 완료 범위를 정의.
테스팅은 특정한 또는 양적으로 많은 목표들을 가진다. 테스팅 목표가 달성될 때, 즉 100% 성공적으로 수행된 시스템이 100% 기능적 커버리지를 만족하는 테스팅이 수행될 때 완성.

2. 테스트 케이스 디자인.
테스트 케이스들은 4 개 특징에 의해 정의된다: 테스트, 데이터, 입력, 시스템의 초기 상태

3. 테스트 케이스를 생성.
테스트 케이스를 만들기 위해 다음과 같은 부분이 필요하다: 필수 데이터 생성과 테스팅을 지원하기 위한 컴포넌트.

4. 테스트를 수행.
문서결과와 테스트된 현재 시스템에 맞서 테스트 케이스 단계를 수행.

5. 테스트 결과를 검증.
테스터들은 테스트 결과의 2 가지 다른 타입을 검증하기 위한 책임: 기대된 결과인가? 테스트 케이스들이 테스트 완성 범위를 만나는가?

6. 테스트 커버리지를 검증.
각 테스트의 성공적인 수행에 의해 성취된 기능적 테스트 커버리지를 추적.

7. 테스트 라이브러리를 관리.
테스트 관리자는 현재 테스트된 프로그램과 테스트 케이스들 사이에 관계쉽을 유지한다. 테스트 관리자는 테스트가 무엇을 수행하였는지 안 하였는지, 수행된 테스트가 통과인지 실패인지의 추적을 유지.

1, 2, 그리고 6 은 요구사항 기반 테스팅에 의해 해당한다. 남겨진 4 가지 활동은 테스트 수행의 상태를 추적하는 테스트 관리 도구에 해당한다.

2.2 테스트 가능한 요구사항을 통한 생명주기

그림 1 은 테스트 가능한 요구사항 생명주기를 나타낸다[9]. 테스트 가능한 요구사항 생명주기는 요구사항 기반 테스팅 프로세스와 전체 소프트웨어 개발 생명주기를 통해 통합된다.

요구사항은 완성되자마자 테스트 된다. 디자인 또한 완성되자마자 요구사항이 디자인에 만족할 만한 특성에 의해 테스트 된다. 디자인 후에 코드가 구조화되고 리뷰 되자마자, 그것은 테스트된다. 이와 같은 테스팅은 요구사항 단계에서부터 시작되기 때문에 대부분 오류는 코드의 실제 테스트 전 발견된다.

이는 즉각적인 방법론이다. 유저 매뉴얼과 훈련 자료들은 즉시 개발될 수 있다. 테스트 가능한 요구사항 생명주기는 다음과 같이 요약된다. 테스팅은 개발과 함께 병행해서 수행된다. 그래서 코드가 인도되었을 때 더 놀라운 결과를 얻게 된다.

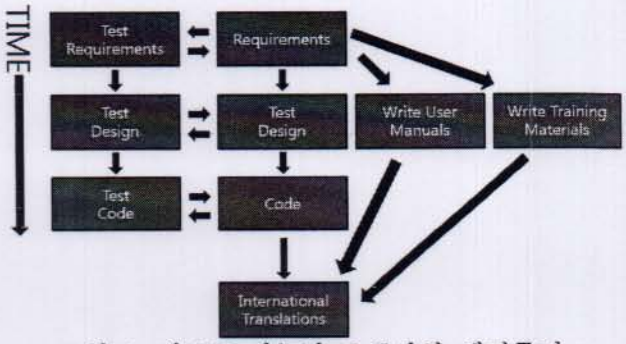

그림 1. 테스트 가능한 요구사항 생명주기

3. 테스트 유닛을 이용한 요구사항 기반 테스팅

3.1 요구사항 기반 테스팅을 위한 절차

그림 2 는 요구사항 기반 테스팅을 위한 전체적인 절차이다. 요구사항 기반 테스팅을 위해서 선행하는 것은 도매인 분석과 비즈니스 목표를 분명히 하는 것이다. 도매인 분석을 통해서 전체 시스템에 대한 명확한 이해와 비즈니스에 대한 목표를 선정해야 한다. 비즈니스 목표기반으로 테스팅에 목표를 세울 수 있다. 이러한 활동이 필수적인 이유는 소프트웨어 테스팅이 매우 광범위한 분야이기 때문이다. 그래서 목표가 분명하지 않은 상태에서 테스팅을 하게 된다면, 무분별한 테스팅으로 이어지고, 그에 대한 결과는 시간과 비용에 낭비로 이어질 수 있기 때문 이다. 하지만 이 논문에서는 더 이상 도매인 분석과 비즈니스 골에 대해서는 제외한다. 아래 그림에서 점선으로 표시한 부분만 논의한다. 분석과 목표에 대한 선택을 통해 요구사항에 대한 분석단계를 시행한다. 요구사항은 소프트웨어 개발 주기에 첫 단계로, 전체 개발 주기 동안 가장 critical 한 영향을 끼친다. 요구사항 분석 단계에서 전체 소프트웨어 오류 중 대부분이

발생한다.

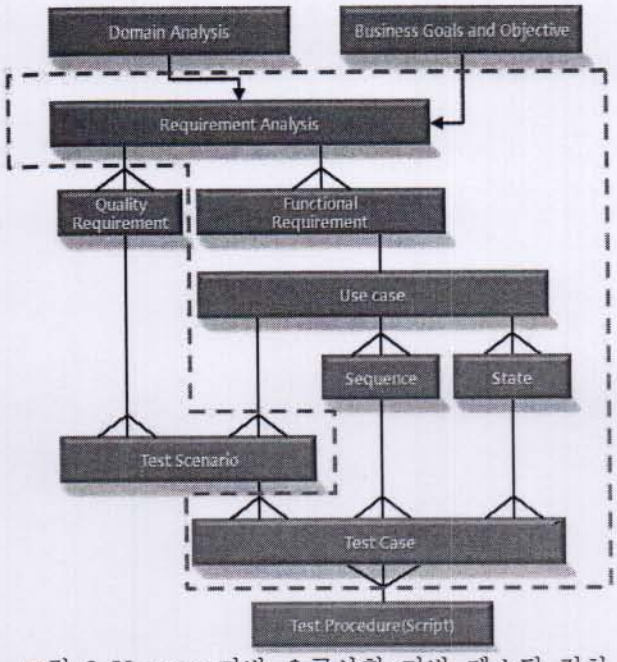

그림 2. Use case 기반 요구사항 기반 테스팅 절차

선행 단계에서 정확한 식별을 통해 요구사항을 분석을 한다면, 소프트웨어 테스팅 시간과 비용, 인력을 줄일 수 있다. 요구사항 분석에서 기능적인 요구사항과 비 기능적인 요구사항을 식별한다. 기능적인, 비 기능적인 요구사항은 기존의 요구사항 분석을 통해 1:1 혹은 1:n 의 관계를 가질 수 있다.

기능적인, 비 기능적인 요구사항에 대한 식별 방법은 생략한다. 기능적인 요구사항으로부터 유스케이스를 추출한다. 유스케이스는 사용자가 시스템과 직접적으로 인터랙션 하는 사항들에 대한 명세이다. 유스케이스와 비 기능적인 요구사항에서 1:1 또는 1:n 관계를 가지는 테스트 시나리오를 추출한다.

이런 테스트 시나리오는 유스케이스를 통해 사용자와 인터랙션 하는 시스템에 대한 시나리오와 비 기능적인 요구사항으로부터 시스템에 개발에서 놓칠 수 있는 시나리오를 포함하기 때문에, 이상적인 테스트 시나리오를 작성할 수 있다. 또한 유스케이스로부터 시퀀스와 스테이트를 생성한다. 테스트 시나리오와 시퀀스, 스테이트로부터 1:1 또는 1:n 관계를 가지는 테스트 케이스를 추출한다. 이 논문에서는 시퀀스와 스테이트를 가지고 테스트 케이스를 추출하는 경우만 언급하겠다. 테스트 케이스는 실질적으로 테스트를 수행하는 세부적인 사항들을 표현한다. 이 테스트 케이스들을 통해서 테스터들은 시스템에 대한 테스팅을 수행하며, 시스템에 대한 문제 발생 시에도 테스트 케이스를 통해서 오류를 찾아낸다. 최종적인 테스트 스크립트를 작성한다. 이와 같은 전체적인 과정이 요구사항 기반으로 테스팅을 수행하는 것이다.

3.2 제안한 테스트 유닛과 테스트 케이스 템플릿

표 1. 테스트 유닛 정의

Test Unit	DEFINITION
Method	한 메시지에 응답하는 객체에 의해서 테스트 되어지는 메소드
Reusable Pattern	특별한 객체 패턴 내에 테스트되는 메소드의 흐름
State	상태 모델에 일치하는 특정한 상태에 반복적으로 테스트 되어지는 메소드의 흐름
MLU(Maximum Linear Unit)	특정한 선택에 의해서 반복적으로 테스트 되어지는 메소드의 흐름
Dialogue	액터에 입력과 액터에 응답에 의해 반복적으로 테스트 되어지는 메소드의 흐름

표 1 의 테스트 유닛(Test Unit)는 1999 년에 정의한 "Design Unit" [10] 개념을 기반으로 재 정의한 것이다. 테스트 유닛(Test Unit)는 기존 테스트 케이스의 기본 형식[input, condition, output]에 대한 단위화를 위해 세분화 한 것이다. 이 정의는 오직 Use Case approach 에서만 적용 가능하다.

1) Method`s Test Unit

-한 메시지에 응답하는 객체에 의해서 테스트 되는 메소드

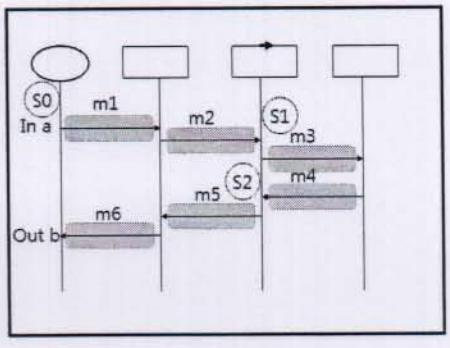

그림 3. 시퀀스 다이어그램에 대한 'Method'의 테스트 유닛

그림 3 은 시퀀스 다이어그램을 간단히 도식화한 그림이다. 이 그림에서 'Method'의 테스트 유닛은 m1, m2, m3 과 같은 객체에서 발생시키는 간단한 메시지이다. 이 메시지는 입력과 예상된 결과를 가진다. 메시지가 컨트롤 객체에 전달되면, 선 상태와 후 상태, 입력을 고려한 상태의 변화로 다양한 기대 값을 가질 수 있다. 그렇기 때문에 이를 통해 테스트 케이스 생성이 가능하다.

표 2. 'Method'에 대한 테스트 케이스 템플릿

테스트 케이스	테스트 메소드 이름	테스팅 타입	선상태	입력 값	후상태	기대된 결과
tc1	m1	Input	S0	In a	None	m2
tc2	m2	Input	None	In a	S1	m3
tc3	m3	Condition	S1	In a	None	m4
tc4	m4	Condition	None	None	S2	m5
tc5	m5	Output	S2	None	None	m6
tc6	m6	Output	None	None	S0	Out b

표 2 는 시퀀스 다이어그램 내에서 'Method'의 테스트 유닛에서 파생되는 테스트 케이스 템플릿을 나타낸 표이다. 이 템플릿은 그림 3 에서 나타낸 시퀀스

다이어그램 내에 메시지에 대한 테스트 케이스이다. 'Method' m1 은 입력으로 In a 를 가진다. 메시지가 전달되었을 때, 초기 상태는 S0 을 나타낸다. m1 이 메시지를 인터페이스 객체에 전달되기 때문에 직접적으로 컨트롤 객체에 전달되지 않는다. 그렇기 때문에 상태의 변화는 없다. 후 상태는 None 이라고 한다. m1 의 결과로써 메시지 m2 를 발생시킨다. TC1 은 테스트해야 하는 테스팅 타입이 시퀀스 다이어그램 내에서 input 요소이기 때문에 테스팅 타입은 Input 이다. 최종적으로 테스트 케이스 TC1 은 위의 표와 같은 형태를 가진다. 나머지 테스트 케이스들도 이와 같은 형식으로 추출 가능하다.

2) Reusable Pattern`s Test Unit
-특별한 객체 패턴 내에 메소드의 흐름

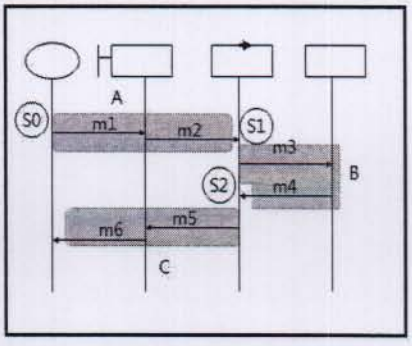

그림 4. 시퀀스 다이어그램에 대한 'Reusable Pattern'의 테스트 유닛

그림 4 는 'Reusable Pattern' 테스트 유닛에 대해 도식화 한 그림이다. 이 그림은 Vending Machine 으로 설명하고 있다. 메시지 m1 과 메시지 m2 는 항상 순차적으로 수행된다. 또한 메시지에 대한 입력과 출력 기대 값은 동일하기 때문에 하나의 Pattern 이라고 정의하는 것이 가능하다. 메시지 m3 과 m4 는 입력된 결과에 대한 출력을 전달하기 때문에 하나의 Pattern 이라고 말할 수 있다. 이와 같은 방법으로 'Reusable Pattern'의 테스트 유닛을 정의한다. 그리고 정의한 Pattern 은 패턴 내에서 메시지들을 동일한 입력과 기대 값, 선상태, 후상태를 가지기 때문에 이를 통해서 Methods 보다 추상화된 테스트 케이스를 생성할 수 있다.

표 3. 'Reusable Pattern'에 대한 테스트 케이스 템플릿

테스트 케이스	테스트 패턴 이름	테스팅 타입	선상태	입력 값	후상태	기대된 결과
TC1	A	Input	S0	Coin	S1	B
TC2	B	Condition	S1	Coin	S2	C
TC3	C	Output	S2	None	S0	Notify

표 3 은 시퀀스 다이어그램 내에서 'Reusable Pattern'의 테스트 유닛에서 파생되는 테스트 케이스 템플릿을 나타낸 표이다. 이 템플릿은 그림 4 에서 나타낸 시퀀스 다이어그램 내에 메시지에 대한 테스트 케이스이다. Pattern A 는 입력으로 Coin 을 가진다. 메시지가 전달되었을 때, 초기 상태는 S0 을 나타낸다. A 가 메시지를 인터페이스 객체에 전달하고 또 직접적으로 컨트롤 객체에 전달하기 때문에 후 상태는 S1 이다. A 의 결과로서 Pattern B 를 발생시킨다. TC1 은 테스트해야 하는 테스팅 타입이 시퀀스 다이어그램 내에서 input 요소이기 때문에 테스팅 타입은 Input 이다. 최종적으로 테스트 케이스 TC1 은 위의 표와 같은 형태를 가진다. 나머지 테스트 케이스들도 이와 같은 형식으로 추출 가능하다.

3) State`s Test Unit
-상태 모델에 일치하는 특정한 상태에 반복적으로 테스트 되는 메소드의 흐름

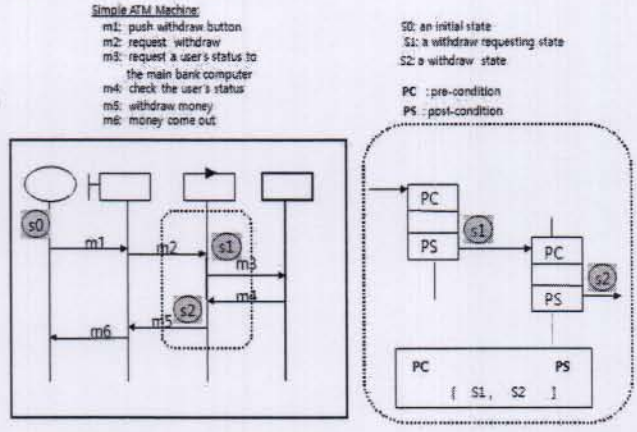

그림 5. 시퀀스 다이어그램에 대한 States 테스트 유닛

그림 5 는 'State'의 테스트 유닛에 대해 도식화 한 그림이다. 이 그림은 ATM Machine 에 대한 출금을 예로 설명하고 있다. 아무런 메시지가 발생하지 않는 상태는 initial 상태이다. 메시지 m1 이 발생할 때는 아무런 상태의 변화를 가지지 않는다. 하지만 메시지 m2 또는 m4 는 컨트롤 객체에 대한 상태를 변화시키기 때문에 컨트롤 객체의 상태 변화 시에 다음과 같이 상태 변화를 시퀀스 다이어그램 상에서 표현이 가능하다. 이때, 입력과 기대 값은 선상태와 후상태에 따라 변화하기 때문에 선상태, 입력 값, 기대된 결과, 후상태를 가지고 테스트 케이스를 만들 수 있다.

표 4. 'State'에 대한 테스트 케이스 템플릿

테스트 케이스	테스트 state 이름	테스팅 타입	선상태	입력 값	후상태	기대된 결과
TC1	an initial state	State	None	None	None	None
TC2	a withdraw requesting state	State	S0	m2	S2	m3
TC3	a withdraw state	State	S1	m4	S0	m5

표 4 은 시퀀스 다이어그램 내에서 'State'의 테스트 유닛에서 파생되는 테스트 케이스 템플릿을 나타낸 표이다. 이 템플릿은 그림 5 에서 나타낸 시퀀스 다이어그램 내에 메시지에 대한 테스트 케이스이다. TC1 의 state 는 initial 상태이므로 입력 값이 없다. 따라서 None 이다. 또 메시지가 전달되지 않으므로 선상태가 None 이다. 상태가 변하지 않으므로 후 상태 또한 None 이다. 최종적으로 테스트 케이스 TC1 은 위의 표와 같은 형태를 가진다. TC2 의 state 는 입력 값이 m2 이다. 또 m2 가 전달되기 전 상태는 initial 이므로 선상태는 S0 이 된다. 기대되는 결과인 m3 을 수행하면 상태가 변하므로 후상태는 S2 이다. 최종적으로 테스트 케이스 TC2 는 위의 표와 같은 형태를 가진다. 나머지 테스트 케이스들도 이와 같은 형식으로 추출 가능하다.

4) MLU`s Test Unit

-특정한 선택에 의해서 반복적으로 테스트 되는 메소드의 흐름

- A MLU(Maximum Linear Unit) is the same as the dialogue test unit. If no choice branch/nodes exist except for the actor.

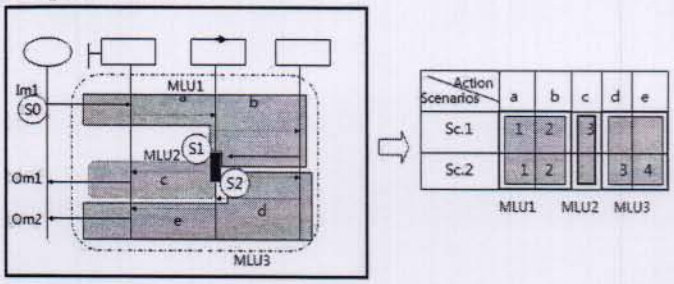

그림 6. 시퀀스 다이어그램에 대한 'MLU'의 테스트 유닛

그림 6 은 'MLU'의 테스트 유닛에 대해 나타낸 그림이다. 이 그림에서 각 메시지는 선택에 의한 분기가 나타나기 전까지를 하나의 'MLU'라고 나타낸다. M1 은 입력 Im1 이 Method 에 순차적인 흐름에 따라 발생하여, 선택이 발생하기까지의 결과이다. M2 는 입력된 Im1 에 의해 Om2 라는 선택을 발생시킨다. M3 은 입력된 Im1 의 또 다른 선택으로 인해서 발생할 수 있는 기대 값 Om2 를 표현한다. 이와 같은 'MLU'는 명백한 입력과 기대 값, 선상태, 후상태를 가지기 때문에 테스트 케이스 생성이 가능하다.

표 5. 'MLU'에 대한 테스트 케이스 템플릿

테스트 케이스	테스트 MLU 이름	테스팅 타입	선상태	입력 값	후상태	기대된 결과
TC1	M 1	MLU	S0	Im 1	S1	M 2 || M 3
TC2	M 2	MLU	S1	None	S0	Om1
TC3	M 3	MLU	S2	None	S0	Om2

표 5 는 시퀀스 다이어그램 내에서 'MLU'의 테스트 유닛에서 파생되는 테스트 케이스 템플릿을 나타낸 표이다. 이 템플릿은 그림 6 에서 나타낸 시퀀스 다이어그램 내에 메시지에 대한 테스트 케이스이다. TC1 은 입력으로 Im1 을 가진다. 메시지가 전달되었을 때, 초기 상태는 S0 을 나타낸다. M1 이 메시지를 인터페이스 객체에 전달하고 또 직접적으로 컨트롤 객체에 전달하기 때문에 후상태는 S1 이다. M1 의 기대되는 결과로서 M2 또는 M3 을 발생시킨다. 최종적으로 테스트 케이스 TC1 은 위의 표와 같은 형태를 가진다. TC2 와 TC3 은 컨트롤 객체에서 두 가지 path 로 파생되어 각각 Om1 과 Om2 로 기대결과가 다르다. 최종적으로 테스트 케이스는 위의 표와 같다.

5) Dialogue`s Test Unit

-액터에 입력과 액터에 응답에 의해 반복적으로 테스트 되는 메소드의 흐름

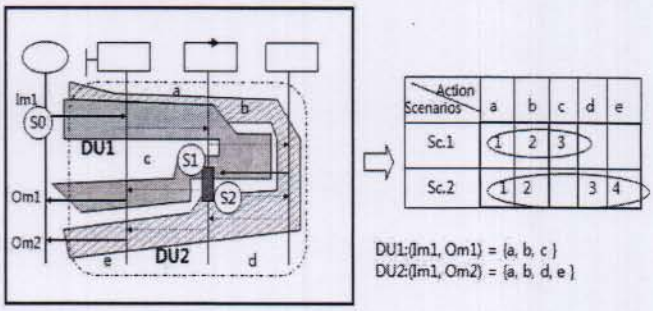

그림 7. 시퀀스 다이어그램에 대한 'Dialogue'의 테스트 유닛

그림 7 은 'Dialogue'의 테스트 유닛을 도식화한 그림이다. 두 가지의 'Dialogue'가 존재한다. DU1 은 정상적이 않은 경우에서 발생하는 'Dialogue'로써 사용자의 입력에 대한 시스템의 비 정상적인 응답을 나타낸다. DU2 는 정상적으로 발생하는 'Dialogue' 이다. 사용자의 입력에 대한 정상적인 응답을 나타낸다. 이렇듯 시퀀스 다이어그램 내에서 사용자의 입력을 기준으로 시스템의 응답에 대해 가장 추상적인 레벨에서 시스템에 대한 테스트 유닛을 적용하는 것이 'Dialogue' 이다. 이 'Dialogue'는 선상태, 입력, 기대 값과 후상태를 고려하여, 테스트 케이스 생성이 가능하다.

표 6. 'Dialogue'에 대한 테스트 케이스 템플릿

테스트 케이스	테스트 Dialogue 이름	테스팅 타입	선상태	입력 값	후상태	기대된 결과
TC1	DU 1	Dialogue	S0	Im 1	S0	Om 1
TC2	DU 2	Dialogue	S0	Im 1	S0	Om 2

표 6 은 시퀀스 다이어그램 내에서 'Dialogue'의 테스트 유닛에서 파생되는 테스트 케이스 템플릿을 나타낸 표이다. 이 템플릿은 그림 7 에서 나타낸 시퀀스 다이어그램 내에 메시지에 대한 테스트 케이스이다. TC1 은 입력으로 Im1 을 가진다. 메시지가 전달되었을 때, 초기 상태는 S0 을 나타낸다. DU1 은 메시지를 인터페이스 객체에 전달하고 또 직접적으로 컨

트롤 객체에 전달한 뒤 다시 인터페이스 객체를 통해 initial 상태가 되기 때문에 후상태는 S0 이다. DU 1 의 기대되는 결과로서 Om1 을 발생시킨다. 최종적으로 테스트 케이스 TC1 은 위의 표와 같은 형태를 가진다. TC2 역시 이와 같은 방법으로 추출할 수 있다.

3.3 테스트 유닛과 테스트 케이스에 대한 레벨화

테스트 유닛의 레벨화에 대한 전체적인 개념은 위의 그림 8 과 같다. 하나의 시스템은 1 개 또는 n 개의 서브시스템을 가진다.

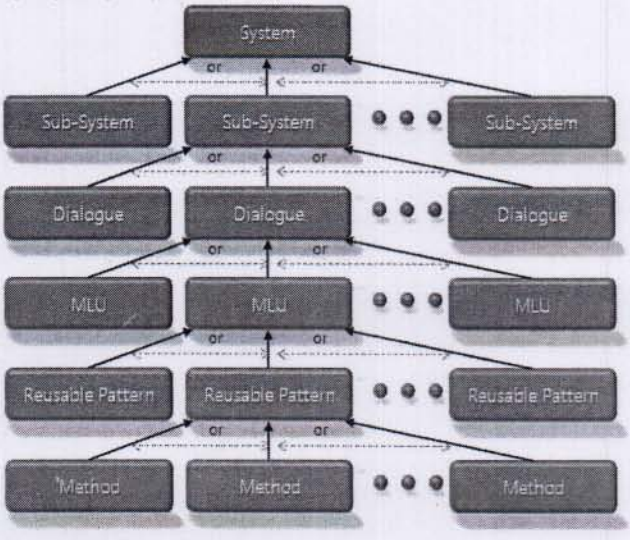

그림 8. 테스트 유닛의 레벨화

하나의 서브시스템은 1 개 또는 n 개의 'Dialogue'의 테스트 유닛을 가진다. 이와 같은 상하 관계로 이루어진 'MLU', 'Reusable Pattern', 'Method'의 테스트 유닛이 존재한다. 시스템과 서브시스템으로부터 추출할 수 있는 시퀀스 다이어그램 내에서 최상위 레벨은 Dialogue 이며, MLU, Reusable Pattern, Method 순서로 테스트 케이스에 대한 레벨을 설정이 가능하였다. 이렇게 설정된 테스트 케이스 레벨은 완전한 테스트 케이스가 추출된 후에 효율적인 테스팅에 초석이 된다.

4. 적용사례

ATM 에 대한 요구사항 명세, 모델링, 테스트 케이스 추출, 레벨화, 우선순위화를 수행 하였다. 이와 같은 과정을 통해서 모든 가능한 테스트 케이스를 추출할 수 있었으며, 효율적인 테스팅을 실제적인 테스트 케이스로 확인이 가능하다. 하지만 지면상에 문제로 요구사항 명세, 모델링은 생략한다.
테스트 유닛 매트릭스는 앞서 정의한 테스트 유닛들에 대한 테스트 케이스 결과를 표로 작성하여 한눈에 테스트 유닛들을 볼 수 있게 정리한 것이다. 앞서 언급한 바와 같이 하나의 서브시스템은 1 또는 n 개의 테스트 Dialogue 유닛을 가질 수 있다. 하나의 Dialogue 유닛에 포함되는 1 또는 n 개의 MLU, State, Pattern, Method 가 존재한다. 다음에서는 ATM 의 출금에 해당하는 테스트 유닛 매트릭스를 보인다.

표 7. ATM 의 출금 테스트 유닛 매트릭스

Dialogue Test Units					MLU Test Units	State Units	Pattern Test Units	Sequential Test Method Units
D24	D25				MLU 28	S0,S1	P52	Insert Card (2)
						S1,S9	P53	Send Confirming Card(2)
					MLU 29	S2,(S3)(S4)	P54	Check Pin Number(3)
						S3,S5	P55	Request PW(2)
	D25				MLU 30	S4,S9	P56	Send No Accept(2)
						S5,S9	P57	Return Card(8)
		D26	D27	D28	MLU 31	S0,S6	P58	Insert PW(2)
					MLU 32	S7,(S8)(S9)	P59	Check PW Number(4)
						S8,S0	P60	Request Item(2)
			D27		MLU 33	S9,S0	P61	Request Re Insert PW(2)
				D28	MLU 34	S9,S0	P62	Send Error PW(2)
D29					MLU 35	S0,S10	P63	Enter Item(2)
						S10,S0	P64	Request Transaction(2)
	D30				MLU 36	S0,S32	P65	Enter Amount(2)
						S33,S0	P66	Display Entered Amount(2)
D31		D32	D33	D34	MLU 37	S0,S34	P67	Confirm Amount(4)
						S35,(S36)(S39)	P68	Verify User Data(3)
					MLU 38	S36,S0	P69	Display Transaction(2)
						S36	P70	Request Transaction(2)
						S37	P71	Response Transaction(3)
						S37,S0	P72	Pay Amount(2)
						S38,S0	P73	Send Terminate(2)
			D33		MLU 40	S39,S0	P74	Send Cancel(2)
				D34	MLU 41	S40,S0	P75	Send Cancel Withdrawal(2)

표 8. ATM 의 출금 테스트 케이스

테스트 케이스	테스트 메소드 이름	테스팅 타입	선상태	입력 값	기대된 결과	후상태
TC 1	insertCard	input	s0	PinNumber	w2	non
TC 2	insertCard	input	non	PinNumber	w3 & w5	s1
TC 3	sendConfirmingCard_MSG	output	s1	msg1	w4	non
TC 4	sendConfirmingCard_MSG	output	non	msg1	non	non
TC 5	verifyPinNumber	condition	s1	Pinnumber	w6(True) || w7(False)	s2
TC 6	notify	condition	s2	TRUE	w8	s3
TC 7	notify	condition	s2	FALSE	w10 & w12	s4
TC 8	requestPW_MSG	output	s3	msg2	w9	non
TC 9	requestPW_MSG	output	non	msg2	non	non
TC 10	sendNoAccept_MSG	output	s4	msg3	w11	non
TC 11	sendNoAccept_MSG	output	non	msg3	non	non
TC 12	returnCard	output	s5	non	w12	non
TC 13	returnCard	output	non	non	non	non
TC 14	insertPW	input	non	PW	w15	non
TC 15	insertPW	input	non	PW	w16	s6
TC 16	verifyPW	condition	s6	PW	w17(True) || w18(False)	s7
TC 17	notify	condition	s7	TRUE	w20	s8
TC 18	notify	condition	s7	FALSE	w22	s9
TC 19	notify	condition	s7	FALSE	w24 & w26	s9
TC 20	requestItem_MSG	output	s8	msg4	w21	non
TC 21	requestItem_MSG	output	non	msg4	non	non
TC 22	requestReinsertPW_MSG	output	s9	msg5	w23	non
TC 23	requestReinsertPW_MSG	output	non	msg5	non	non
TC 24	sendErrorPW_MSG	output	s9	msg6	w25	non
TC 25	sendErrorPW_MSG	output	non	msg6	non	non
TC 26	returnCard	output	s5	non	w27	non
TC 27	returnCard	output	non	non	non	non
TC 28	enterItem	input	non	inputItem	w29	non
TC 29	enterItem	input	non	inputItem	w30	s10
TC 30	displayEnteredItem	output	s10	enteredItem	w31	non
TC 31	displayEnteredItem	output	non	enteredItem	non	non
TC 32	enterAmount	input	non	inputAmount	w33	non
TC 33	enterAmount	input	non	inputAmount	w34	s11
TC 34	displayEnteredAmount	output	s12	msg7	w35	non
TC 35	displayEnteredAmount	output	non	msg7	non	non
TC 36	confirmAmount	input	non	TRUE	w37	non
TC 37	confirmAmount	input	non	TRUE	w40	s12
TC 38	confirmAmount	input	s12	FALSE	w39	s13
TC 39	confirmAmount	input	non	FALSE	w41	s12
TC 40	verifyUserData	condition	s13	PinNumber, inputAmount	w42(True) || w43(False)	s13
TC 41	notify	condition	s13	TRUE	w43 & w45	s14
TC 42	notify	condition	s13	FALSE	w57 & w58	s16
TC 43	displayTransaction	output	s14	resultItem	w44	non
TC 44	displayTransaction	output	non	resultItem	non	non
TC 45	requestNetworkTransaction	input	s14	transactionList	w46	non
TC 46	requestNetworkTransaction	input	non	transactionList	w47	non
TC 47	requestNetworkTransaction	input	non	transactionList	w48	non
TC 48	responseTransaction	output	non	balance	w49	non
TC 49	responseTransaction	output	non	balance	w50	non
TC 50	requestTransaction	output	non	balance	w51 & w53 & w55	s15
TC 51	payAmount	output	s15	inputAmount	w52	non
TC 52	payAmount	output	non	inputAmount	non	non
TC 53	returnCard	output	s5	non	w54	non
TC 54	returnCard	output	non	non	non	non
TC 55	sendTerminate_MSG	output	s5	msg8	w56	non
TC 56	sendTerminate_MSG	output	non	msg8	non	non
TC 57	sendCancel_MSG	output	s16	msg9	w58	non
TC 58	sendCancel_MSG	output	non	msg9	non	non
TC 59	returnCard	output	s5	non	w60	non
TC 60	returnCard	output	non	non	non	non
TC 61	sendCancelWithdrawal_MSG	output	s12	msg10	w62	non
TC 62	sendCancelWithdrawal_MSG	output	non	msg10	non	non
TC 63	returnCard	output	s5	non	w63	non
TC 64	returnCard	output	non	non	non	non

표 7 은 출금에 대한 테스트 유닛 매트릭스를 표현하였다. 출금에 관련된 Dialogue 는 총 11 개, MLU 13 개, State 19 개, Reusable Pattern 24 개, Method 64 개가 존재한다.

표 8 은 출금에 대한 모든 테스트 케이스이다. 이 테스트 케이스를 모두 수행하려면, 총 64 개의 테스트 케이스를 수행해야 한다.

하지만 테스트 매트릭스를 이용한다면, 효율적인 테스팅을 할 수 있다. 이 방법은 그림 10(a, b) 에서 표현하였다. 일반적으로 시스템에 문제가 발생하였다면, 문제에 관련된 모든 테스트 케이스를 고려해야 한다. 하지만 테스트 유닛 매트릭스를 사용한다면, 시스템 전체에 대한 테스팅에 있어서 High-level 에서 Detail-level 로 테스팅이 가능할 것이다.

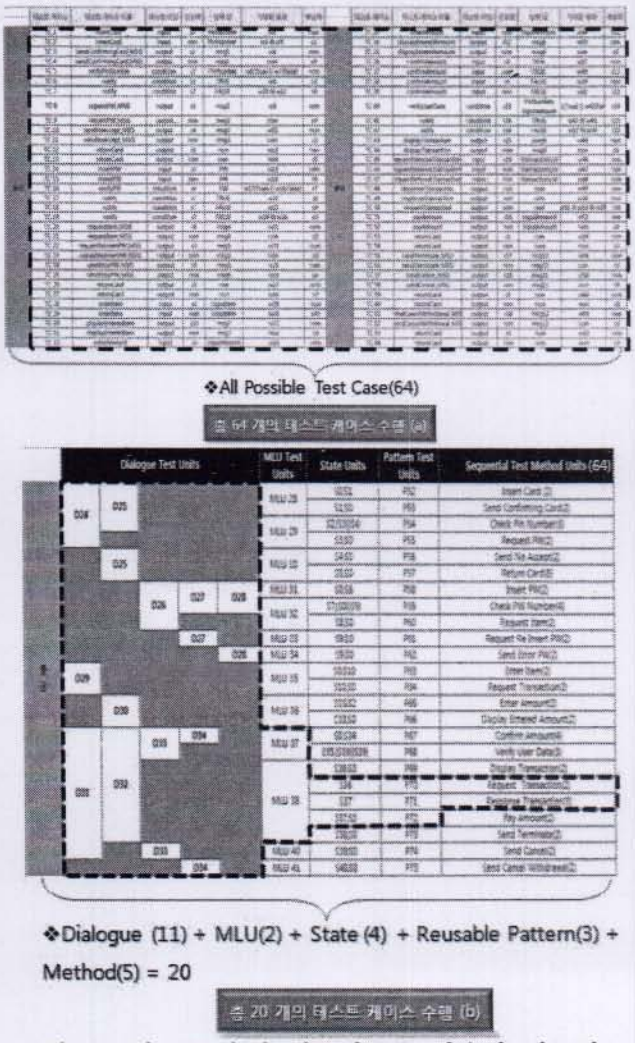

그림 10. 테스트 유닛 매트릭스를 이용한 테스팅

ATM 에서 문제 발생 시에 기존에 테스트 케이스를 통한 테스팅 방법은 모든 테스트 케이스를 수행해야 한다. 출금에 관한 문제 발생 시에 64 개의 테스트 케이스를 모두 수행하여야 한다. 제안된 방법은 정의된 테스트 유닛을 통해 Dialogue D24~D34 까지 테스팅을 수행한다. 그 중 D32 에서 문제가 발생하면, D32 가 포함하는 MLU 37, 38 에 대한 더 세부적인 테스팅을 진행한다. MLU 38 에서 문제가 발생하면, MLU 38 이 포함하고 있는 State, Reusable Pattern 에 대한 테스팅을 수행한다. 이때, Pattern P70, 71, 72 중에서 P71, 72 문제가 발생하였다면, P71, 72 가 포함하고 있는 Method 5 개를 테스팅 한다. 5 개의 Method 를 테스팅 하여 올바르게 동작한다면, ATM 에 출금에 관련된 문제를 해결했다고 말할 수 있다.

이와 같은 매트릭스로 테스팅을 수행한다면, 기존의 64 개의 테스트 케이스를 모두 수행하는 것이 아니라, 20 개의 테스트 케이스를 수행하여도 똑같은 결과를 얻을 수 있는 것이라 생각한다. 이로써, 최소의 테스트 케이스로부터 최대의 커버리지를 만족할 수 있는 것이다. 위의 과정을 통해서 최소의 테스트 케이스를 확보 하였다.

5. 결론 및 향후 연구

임베디드 시스템의 발달에 따라 임베디드 소프트웨어의 복잡성은 날로 증가해지고 있다. 그로 인해 테스팅이 범위는 더 광범위해지고 있다. 이 때문에 테스트 비용과 시간을 증가함으로써 완벽한 테스팅이 어렵다.

본 논문은 제안한 테스트 유닛을 이용한 요구사항 기반 테스팅을 통해 테스트 비용과 시간을 줄이고자 한다. 유스케이스 패러다임 상에서 요구사항을 분석, 요구사항 기반 모델링을 통해 유스케이스, 시퀀스, 스테이트와 같은 세부적인 모델링을 도출하고, 정의한 테스트 유닛들과 테스트 케이스 템플릿을 통해 테스트 케이스를 추출한다. 시퀀스 다이어그램 상에서 테스트 유닛을 기반으로 테스트 케이스에 대한 레벨화를 수행하였다. 현재 가중치를 통한 테스트 케이스에 대한 우선순위화를 정의하여 도구화가 진행중이다. 이를 통해 앞으로 최소의 테스트 케이스로부터 최대의 테스트 커버리지의 효율적인 테스팅을 실행 방법을 제안하려 한다.

또한, 향후 연구과제로 유스케이스에 Goal 을 도입하여, 좀 더 명확하고 정량적인 분석을 통한 모델링부터 테스트 케이스까지 연관할 수 있는 Goal Oriented Requirement Based Testing 에 대해 연구할 것이다.

참고문헌

[1] 김홍남, 박승민, 김두현, "임베디드 소프트웨어 최근 기술 동향", 정보과학회지, 제 24 권, 제 8 호, pp. 5-11, 2006. 8

[2] 이선열, 배정호, 채홍석, "테스트 데이터 자동 생성에 적용된 유전 알고리즘에 대한 비교", 한국정보과학회, 제 36 권, 제 1 호(B), pp.

51~56, 2009.6

[3] 권상훈, 한혁수, 우치수, "객체지향 소프트웨어의 테스팅 전략에 관한 연구", 한국정보과학회, 제 22 권, 제 2 호(B), pp. 803~1520, 1995.10

[4] 박현상, 최경희, 정기현, "요구사항 기반 테스팅의 유효한 테스트 케이스 판별 전략", 한국정보과학회, 제 36 권, 제 1 호(B), pp. 90~95, 2009.6

[5] 한정희, "소프트웨어 에러, 수백억 달러 손해 낸다", KOTRA, 2002

[6] Steven R. Rakitin, "Software Verification And Validation", 1997

[7] 지은경, 유준범, 박수현, 차성덕, "Function Block Diagrams에 대한 제어 및 데이터 흐름 테스팅," 소프트웨어공학회지, Vol. 18, No.1, pp.3-19, 2005

[8] Gary E. Mogyorodi, "What Is Requirements-Based Testing?", CROSSTALK The Journal of Defense Software Engineering, 2003

[9] G Mogyorodi, "Requirements-based testing: an overview", computer.org, tools, 2001

[10] Youngchul Kim, C. Robert Carlson, "Scenario Based Integration Testing for Object-Oriented Software Development," ats, pp.283, Eighth Asian Test Symposium (ATS'99), 1999